Evaluating the Impact of Pruning on Transformer Models: A Comparative Study

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Master of Technology
in
Computer Science & Engineering

by

Sourabh Sathe (MT23MCS003)

under the guidance of

Dr. Ashish Tiwari

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

Evaluating the Impact of Pruning on Transformer Models: A Comparative Study

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Master of Technology
in
Computer Science & Engineering

by

Sourabh Sathe (MT23MCS003)

under the guidance of

Dr. Ashish Tiwari

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

©Visvesvaraya National Institute of Technology (VNIT) 2025

Department of Computer Science & Engineering

Visvesvaraya National Institute of Technology, Nagpur

Declaration

I, **Sourabh Sathe**, hereby declare that this dissertation titled **"Evaluating the Impact of Pruning on Transformer Models: A Comparative Study"** is carried out by me in the Department of Computer Science & Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree at this or any other Institution.

Sourabh Sathe

Date: 30/5/2025

Certificate

This to certify that the dissertation titled **"Evaluating the Impact of Pruning on Transformer Models: A Comparative Study"** is submitted by **Sourabh Sathe** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science & Engineering**, VNIT Nagpur. The work is comprehensive, complete, and fit for final evaluation.

Dr. Ashish Tiwari

Assistant Professor

Department of Computer Science & Engineering

VNIT, Nagpur

Dr. M.P. Kurhekar

Head, Department of Computer Science & Engineering

VNIT, Nagpur

Department of Computer Science & Engineering

Visvesvaraya National Institute of Technology, Nagpur

Non-Plagiarism Certificate

Certified that M. Tech Dissertation titled **"Evaluating the Impact of Pruning on Transformer Models: A Comparative Study"** submitted by **Sourabh Sathe**, Enrollment No. **MT23MCS003**, ID No. **30087** has been checked for plagiarism using TURNITIN software and the overlap is found within prescribed limits. A summary of the report is given below.

Sourabh_MTECH_THESIS_Transformer_Pruning (1).pdf

ORIGINALITY REPORT

18%	12%	10%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

Sourabh Sathe

Enrollment No: MT23MCS003

Dr. Ashish Tiwari

Assistant Professor

Department of Computer Science & Engineering

VNIT, Nagpur

ACKNOWLEDGEMENT

I would like to take this opportunity to express my profound gratitude to all those who have contributed to the successful completion of my M.Tech dissertation.

First and foremost, I am deeply thankful to my supervisor, Dr. Ashish Tiwari, for his constant guidance, support, and encouragement. His wealth of knowledge, thoughtful insights, and patient mentorship have played a pivotal role in shaping the direction and quality of this work. I feel truly privileged to have worked under his supervision.

My sincere thanks are also extended to the esteemed faculty of the Department of Computer Science & Engineering, Visvesvaraya National Institute of Technology, Nagpur, for their continuous support and for fostering an environment that encourages academic rigor and intellectual curiosity. Their valuable inputs, whether through coursework or discussions, have enriched my learning and significantly contributed to my research.

I am also immensely grateful to my family and friends, whose unwavering support, understanding, and motivation have been my strength throughout this journey. Their belief in me has been a source of inspiration during moments of challenge and growth.

Thank you all.

Sourabh Sathe

ABSTRACT

Self-Attention plays a crucial role in machine learning. At the heart of a transformer, the attention mechanism is used to build complex relationships between words in sentences. A transformer uses a sequence of layers with multiple attention heads. Studies have shown that not all attention heads contribute useful information. Some heads can be pruned with little to no effect on the accuracy of the model. However, pruning can have a huge impact on the computational performance such as reduction in inference time and memory usage. Many existing optimization techniques employ pruning with a focus on attention heads while little to no optimization in the FFN layers. The key challenge is identifying which heads and neurons should be pruned. In this project, we employ our own dual-step pruning algorithm that optimizes both attention heads and FFN neurons by pruning them to evaluate the impact of pruning. Through experimentation, we assess the performance of multiple transformers with multiple datasets. We further conclude if pruning is really beneficial and explain its real-life implications.

LIST OF FIGURES

3.1 Components of Transformer 7
3.2 BERT Architecture 14
3.3 Self-Attention 15
3.4 Linear Transformation Matrix derived from Q and K 17
3.5 Linear Transformation by V 17
3.6 Multi-Head Attention 19
3.7 Multi-head concatenation 20
3.8 Multi-head Attention output 20
3.9 Heat maps of attention heads extracted from the transformer 22

4.1 Flowchart of steps used in the dual-step pruning algorithm. 24
4.2 A list of tasks in GLUE benchmark 25
4.3 Algorithm Implementation Workflow for BERT model 31

5.1 Output of HuggingFace Transformer Benchmark Tool 34
5.2 Implementation Workflow of Experiment 1 and 2 36
5.3 Implementation Workflow of Experiment 3 and 4 37

6.1 Heads ranked by importance scores 38
6.2 Training Loss vs No. of Training Steps 41

LIST OF TABLES

3.1 Tokenization example 8

3.2 Comparison of Q, K, and V matrices 17

3.3 Comparison of Multi-head Attention Types in Transformer Architecture . 21

6.1 Comparing Baseline vs. Pruned `bert-base-uncased` model on SST-2 39

6.2 Inference Time Comparison of Baseline vs. Pruned model on SST-2 . . . 40

6.3 Memory Usage Comparison of Baseline vs. Pruned model on SST-2 . . . 40

6.4 Comparison of accuracies between baseline and pruned `bert-base-uncased` model after fine-tuning on various GLUE tasks. 40

NOMENCLATURE

Hyperparameters

α	Gradient-Entropy Weighting Factor
μ_h	Head Masking Fraction
μ_n	Neuron Quantile Threshold
τ	Head Masking Threshold

Acronyms

API	Application programming interface
BERT	Bidirectional Encoder Representations from Transformers
FFN	Feed Forward Network
GLUE	General Language Understanding Evaluation
GPT	Generative Pre-trained Transformer
IMP	Iterative Magnitude Pruning
KD	Knowledge Distillation
LLM	Large Language Models
LoRa	Long Range
LTH	Lottery Ticket Hypothesis

MLM	Masked Language Model
MoE	Mixture of Experts
MRPC	Microsoft Research Paraphrase Corpus
NAT	Negative Assistant Training
NLP	Natural Language Processing
NSP	Next Sentence Prediction
NVML	NVIDIA Management Library
RNN	Recurrent Neural Network
RTE	Recognizing Textual Entailment
SQuAD	Stanford Question Answering Dataset
SST-2	Stanford Sentiment Treebank
WNLI	Winograd Natural Language Interface

LIST OF PUBLICATIONS

[1] Sourabh Sathe, Purvaja Kale, Ashish Tiwari, *Evaluating the Impact of Pruning on Transformer Models: A Comparative Study*

Accepted for Full Paper Presentation in *Seventh International Conference on Computational Intelligence in Communications and Business Analytics (CICBA 2025)*

To be published in *Communications in Computer and Information Science (CCIS), Springer (Scopus and DBLP Indexed)*

INDEX

ABSTRACT **i**

LIST OF FIGURES **ii**

LIST OF TABLES **iii**

NOMENCLATURE **iv**

LIST OF PUBLICATIONS **vi**

INDEX **vii**

1 INTRODUCTION **1**

1.1 Overview 1

1.2 Objectives 2

1.3 Thesis Outline 2

2 RELATED WORK **4**

2.1 Lottery Ticket Hypothesis 4

2.2 Existing Optimization Techniques 4

2.2.1 Sparsification 4

2.2.2 Knowledge Distillation 5

2.2.3 Quantization 6

2.2.4 Mixture of Experts 6

2.3 Limitations 6

3 PRELIMINARY STUDY **7**

3.1 Transformer Architecture 7

3.1.1 BERT 11

3.2 Attention Mechanism 15

3.2.1 Self-Attention 15

3.2.2 Multi-head Attention 18

3.3 Need for Pruning 21

3.3.1 Redundancy in Attention layers 21

3.3.2 Redundancy in FFN layers 22

4 METHODOLOGY **23**

4.1 Proposed Work 23

4.2 GLUE Benchmark 25

4.3 Importance Scores 26

4.4 Algorithm 26

5 IMPLEMENTATION **32**

5.1 Setup 32

5.2 Experiments 32

5.3 Performance Benchmarking Tools 33

6 RESULTS AND OBSERVATIONS **38**

6.1 Metric Comparison 38

6.2 Question Answering 40

7 CONCLUSION AND DISCUSSION **43**

REFERENCES **45**

Chapter 1

INTRODUCTION

1.1 Overview

The transformer architecture was first proposed by Vaswani et al. [1] in the paper *"Attention Is All You Need"*, which has, in turn, greatly shaped deep learning and natural language processing (NLP). Self-attention as a mechanism enables transformers to establish sophisticated dependencies between words in a sequence regardless of their relative position in the sentence. While transformers achieve remarkable performance on a wide range of NLP tasks, they are computationally very expensive. These multi-head attentions add significantly to the memory footprint and inference latency of the model. Research has shown that all attention heads are not equally important for model performance. Some heads encode redundant or less useful information, leading to excessive consumption of memory and computation with no benefit at all. It led to studies in model pruning where redundant components are targeted to be removed for more efficiency without much loss in accuracy. Many researchers have written model pruning algorithms with a primary focus on pruning attention heads. However, FFN layers, which constitute a large portion of the model parameters, also introduce redundant computations. Furthermore, whether pruning is really beneficial in optimizing the memory and inference time of the transformer model remain a question to be answered. While one pruned transformer model may work extremely well for a certain NLP task, it may fail for another. To address these queries, we perform rigorous experimentation on multiple transformer models with multiple datasets by using our own **Dual-step pruning algorithm** that optimizes both attention layers and FFN layers, ensuring a balanced and effective reduction in model size and inference time.

1.2 Objectives

This project aims to answer the following key questions:

- Which attention heads can be pruned without affecting model accuracy?
- How does pruning attention heads and FFN layer neurons impact the overall performance of transformers?
- What are the trade-offs between inference time, memory, and accuracy?
- How does pruning affect different transformer models across various NLP tasks and datasets?
- Does pruning affect the output on real-life question-answering task?

1.3 Thesis Outline

- **Chapter 1** provides an introduction to the project, covering essential background and the motivation behind the study. It highlights the relevance of the chosen topic. The chapter concludes by outlining the key objectives that guide the entire work.
- **Chapter 2** presents a comprehensive literature review, summarizing past and current research efforts focused on optimizing transformer models and large language models (LLM). It also discusses the limitations and gaps identified in these studies.
- **Chapter 3** outlines the preliminary investigations undertaken to build a strong foundational understanding of transformer architectures. This includes a case study on BERT, an in-depth explanation of the attention mechanism, and a discussion on the necessity of pruning in such models.
- **Chapter 4** describes the proposed methodology adopted for the project. It elaborates on the overall implementation workflow, the dataset utilized, the algorithms applied, and the benchmarking metrics used for performance evaluation and comparison.
- **Chapter 5** outlines the implementation setup and presents flowcharts for the various experiments to be conducted.

- **Chapter 6** presents the results obtained through extensive experimentation. It includes comparison tables, training and evaluation graphs, along with corresponding observations for each result. Additionally, it demonstrates the application of the pruned model in a real-world question-answering task.
- **Chapter 7** summarizes the key findings and outcomes of the project, addressing the research questions posed. It also highlights the potential real-world applications of the work and discusses future directions for further improvement and exploration.

Chapter 2

RELATED WORK

2.1 Lottery Ticket Hypothesis

[2] Neural network pruning techniques can reduce the total no. of parameters of a trained model by over 90% and improve storage and performance without compromising accuracy. Standard pruning techniques naturally uncover subnetworks. Actual experiments show that such subnetworks can be as small as 10-20% of the original network's size yet still achieve the same or even better performance than the full model. Following the core ideas of the Lottery Ticket Hypothesis (LTH), recent works have begun to consider its implications for LLMs, especially pruning. For instance, Chen et al. [3] demonstrated that within pre-trained models like BERT, it is possible to identify sparse subnetworks—termed "winning tickets"—that, when trained in isolation, can achieve performance comparable to the full model across various downstream tasks. These findings suggest that significant portions of the attention parameters in LLMs may be redundant, and that pruning can lead to more efficient models without substantial loss in accuracy. This project stems from this fact.

2.2 Existing Optimization Techniques

2.2.1 Sparsification

In the context of Machine Learning and Transformers, sparsification, also known as pruning, refers to the process of removing unnecessary parameters (weights) from a model to reduce its size and complexity, while ideally maintaining performance.

Sun et al. [4] proposed a new, simple but efficient pruning approach, called Wanda

(Pruning by Weights and activations), that is specifically aimed at introducing sparsity into pretrained LLMs. Inspired by the recent finding of emergent large magnitude features in LLMs, their pruning method prunes weights with smallest magnitudes times corresponding input activations on a per-output basis. This method efficiently prunes LLMs to high levels of sparsity with no requirement to adjust the surviving weights.

Jaiswal el at. [5] exhaustively examines induced sparse patterns in several large pretrained vision and language transformers. With the exploding numbers of parameters, Lottery Ticket Hypothesis (LTH) and its extensions, have forfeited their usefulness in sparsifying them because of high computation and memory bottleneck of iterative magnitude pruning's recurring train-prune-retrain cycle that degrades with growing model size.

Rogers at al. [6] focused on the current proposals to improve BERT's architecture, pretraining and fine-tuning. They also discussed approaches to compress BERT model and the area of pruning as a model analysis technique. They proved that BERT can be efficiently compressed with very less accuracy loss, which is desirable for real-world applications.

2.2.2 Knowledge Distillation

Knowledge distillation is a technique where knowledge, typically in the form of prediction probabilities, is transferred from a large, complex "teacher" model to a smaller, more efficient "student" model. This process allows the student model to achieve comparable performance to the teacher while being significantly smaller and faster.

Sanh et al. [7] developed DistilBERT, a compact and efficient alternative to BERT that uses knowledge distillation. It's about 40% smaller and runs 60% faster, yet still preserves around 97% of BERT's language comprehension ability. Their work demonstrated that it's possible to train a versatile language model effectively through distillation, and they backed this up with an ablation study to examine the impact of different components.

Li et al. [8] proposed a novel model specialization framework that can exploit negative data to enhance the distillation of the complex reasoning abilities from LLMs. Specifically, they developed negative assistant training (NAT) approach, where dual LoRA structure is designed to capture knowledge from both positive and negative sides.

2.2.3 Quantization

Quantization is a model compression technique that converts the weights and activations within a large language model from high-precision values to lower-precision ones. Reducing the number of bits required for each of the model's weights or activations leads to a significant decrease in its overall size, consume less memory, require less storage space, and make them more energy-efficient.

Dettmers et al [9] introduced QLoRA, a memory-efficient approach that maintains strong performance while significantly reducing resource usage. They innovated a new data type called 4-bit NormalFloat (NF4), designed to be information-theoretically optimal for weights that follow a normal distribution. They applied double quantization that compressed not just the model weights but also the quantization constants to further lower the overall memory footprint along with paged optimizers for better memory management.

2.2.4 Mixture of Experts

A mixture of experts (MoE) model operates on this principle by dividing a complex task among smaller, specialized networks known as ”experts”. Each expert focuses on a specific aspect of the problem, enabling the model to address the task more efficiently and accurately. By collaborating, these specialists can solve a broader range of problems more effectively than a single generalist.

2.3 Limitations

A common limitation among existing optimization techniques is often focusing exclusively on either attention heads or FFN layers. These methods treat the components in isolation, potentially missing out on optimization opportunities arising from their interdependencies. The techniques are, however, complex when it comes to implementation which is why we choose pruning as the optimization algorithm. Furthermore, they give only a rough idea of the model's real world performance and benefits. The project addresses this limitation by performing rigorous experimentation, analysis by comparisons, thus proving the Lottery Ticket Hypothesis.

Chapter 3

PRELIMINARY STUDY

3.1 Transformer Architecture

Transformer follows an encoder-decoder architecture. The core idea behind the transformer presented by Vaswani et al. [1] is the use of self-attention mechanisms that process each word in relation to all other words in a sentence. This is contrary to traditional methods that process words in sequential order (RNN).

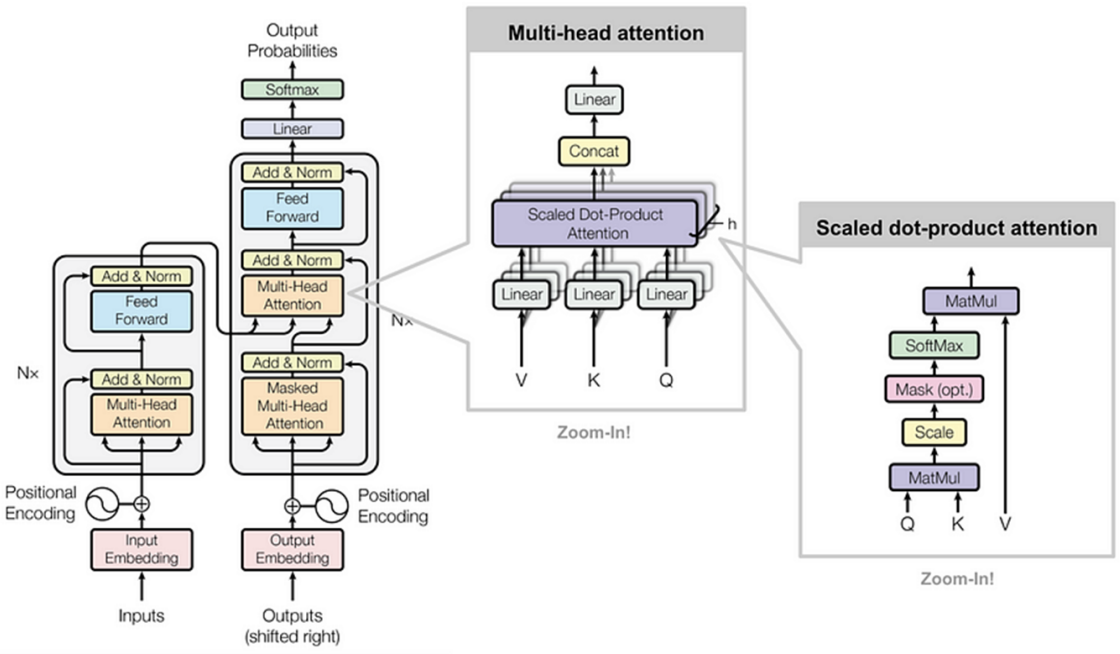


Figure 3.1: Components of Transformer

A transformer is made up of the following components: (Refer figure 3.1)

1. **Tokenization / Encoding**

   Transformers take inputs in the form of tokens. Tokenization consists of breaking

down text into tokens, which can be words or letters, depending on the method used. Tokens help the model understand the actual elements in the input, associating relationships between them. There are mainly 3 types of tokenization methods:

- Character-based Tokenization: Splits text into individual characters.
  Input: `'apple'`
  Output: `['a', 'p', 'p', 'l', 'e']`
- Word-based Tokenization: Splits the sentence into individual words.
  Input: `'an apple a day keeps doctor away'`
  Output: `['an', 'apple', 'a', 'day', 'keeps','doctor', 'away']`
- Subword-based Tokenization: Combines the advantages of both word and character tokenization by breaking words into frequently occurring subword units. This method is used in BERT, GPT, and other transformer models.
  Input: `'unaffordable'`
  Output: `['un', 'afford', 'able']`

Consider the following lowercase dataset:
`"i bought an apple phone. the basket contains an apple and an orange."`

Encoding unique words using word-based embedding and serializing them yields:

Table 3.1: Tokenization example

Word	Token ID
apple	0
phone	1
i	2
an	3
the	4
basket	5
bought	6
contains	7
and	8
orange	9

Vocabulary size is the count of unique words in the corpus.[1] For table 3.1,

$$\text{Vocab Size} = |\text{Set of Unique Words}| = 10$$

2. **Embeddings**

   Embeddings are mathematical depictions of relationships between tokens in a multi-dimensional space. These help the model understand complex relationships and encode them into a single vector. Word embedding vector values are randomly initialized between -1 and 1 (or sometimes between 0 and 1). As training progresses, these values are updated as the Transformer learns the semantic meanings and relationships among words. The Transformer architecture proposed in the original paper uses a 512-dimensional embedding vector for each input word:

   $$\text{Embedding Matrix} \in R^{d_{\text{model}} \times n}, \quad \text{where } d_{\text{model}} = 512, \text{n = no. of tokens}$$

3. **Positional Encoding**

   In natural languages, the arrangement of words determines the meaning of the sentence. Changing the word order can completely alter the meaning.

   Example: "The teacher helped the student." is different from *"The student helped the teacher."* although both sentences have the same unique words.

   While Recurrent Neural Networks (RNNs) naturally account for the sequence of inputs due to their recurrent structure, Transformer models work differently. Unlike RNNs or convolutional models, Transformers process all input tokens simultaneously and independently, without any built-in sense of order. To address this, Transformers rely on positional encoding which is a method that injects information about the position of each word in the sequence. This allows the model to understand the structure and sequence of the input, preserving the relationships between words based on their order.

[1] GPT-3 has a vocabulary size of 50,257 tokens.

Positional Embedding for any token is given as:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \qquad \text{(for even positions)}$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \qquad \text{(for odd positions)}$$

where d_{model} is the dimension of the embedding vector.

The frequency term $\left(10000^{2i/d_{\text{model}}}\right)$ Controls the *wavelength* of sinusoidal functions. Lower dimensions (small i) capture *high-frequency* (local) positional information. Higher dimensions (large i) capture *low-frequency* (global) positional information.

Using both sine and cosine functions ensures that each position gets a unique encoding. Alternating sine and cosine avoids symmetry issues and provides complementary information, as sine and cosine are phase-shifted versions of each other, enriching the positional representation.

4. **Attention Mechanism**

   The attention mechanism is a powerful method that enables models to selectively concentrate on relevant parts of the input sequence when generating each part of the output. By assigning varying weights to different input tokens, the model can emphasize the most important pieces of information, helping it make better predictions. This approach is especially effective in tasks like language translation, where understanding the meaning of a word often requires interpreting its surrounding context. The mechanism will be explained in detail in section 3.2

5. **Adding and Normalization**

   The Add and Norm layer is a type of residual connection. The advantage of this is to facilitate signal propagation in both backward and forward paths which solves the problem of vanishing gradients. Complex networks with deeper layers like transformer suffers from vanishing gradient problems and skip connection helps restore the diminishing gradients by performing the identity mapping.

   During the training process in the transformer, the outputs produced by neurons in a layer after applying activation function to the weighted sum of inputs is called as

activations. The distribution of these activations changes over time due to changes in network parameters (called covariance shift). Normalization mitigate this shift by maintaining stable distribution of activations, which makes the training process smooth.

6. **Feed Forward Neural Network** After normalizing, the matrix is processed through a feed-forward network. A very basic network is to use one linear layer and one ReLU activation function layer.

   The feed-forward network performs two main tasks:

   (a) Non-linear Transformation: While self-attention captures context-aware relationships between tokens, the feed-forward network introduces additional depth by applying non-linear transformations to these representations. This helps the model learn more complex patterns and relationships within the data.

   (b) Feature Enhancement: The feed-forward layer also serves to further process and refine the outputs from the self-attention mechanism. By amplifying key features and minimizing less important ones, it enhances the quality of the representations, making the final outputs more meaningful and informative.

3.1.1 BERT

Traditional language models read text in a sequential manner—either from left to right or right to left—restricting their understanding to the context that comes before a given word. In contrast, BERT (Bidirectional Encoder Representations from Transformers) introduced by Devlin et al. [10] takes a different approach by analyzing the entire sentence at once. It captures both the left and right context of each word simultaneously, rather than processing the text in order.

Example: "The bank is situated on the _____ of the river."

In a unidirectional model, interpreting the blank relies mainly on the words that come before it, which can make it difficult to determine if "bank" refers to a financial institution or a riverbank. However, BERT's bidirectional nature allows it to take into account both

the preceding context (”The bank is situated on the”) and the following context (”of the river”) at the same time. This enables the model to better infer that the word ”bank” in this case refers to a geographical location, showcasing the deeper contextual understanding made possible by a bidirectional approach.

When training language models, setting a clear prediction objective can be tricky. Many models focus on predicting the next word in a sequence—a one-directional method that can limit how well the model understands context.

BERT tackles this issue using two smart training strategies:

1. **Masked Language Model (MLM)**

   During BERT's pre-training, it randomly hides a portion of the words in a sentence (about 15%) and learns to predict those hidden words by analyzing the surrounding context.

   - Hiding Words: Before training, BERT masks some words in a sentence by replacing them with a special token like [MASK].
   - Guessing the Hidden Words: BERT's task is to predict what these masked words are, using the words that appear around them. It's similar to solving a fill-in-the-blank puzzle.

   How BERT Learns:

   - BERT uses a special prediction layer on top of its architecture to guess the masked words.
   - It evaluates how close its predictions are to the actual hidden words by assigning probabilities to possible word choices.
   - The training process focuses mainly on these masked words, helping the model sharpen its understanding of word meanings in context.

2. **Next Sentence Prediction (NSP)**

   BERT is trained to determine whether one sentence logically follows another. It does this by using the output of the special [CLS] token, which is passed through a classification layer and transformed into a 2×1 vector. A SoftMax function then

calculates the probability that the second sentence follows the first in the original text.

How it works during training:

- BERT is given pairs of sentences. In 50% of the cases, the second sentence is the actual next sentence from the original document. In the other 50%, it's a random sentence from the corpus.
- This setup teaches the model to identify whether two sentences are contextually linked or not.

Input Preparation:

- A [CLS] token is added at the start of the first sentence to represent the whole input pair.
- A [SEP] token is placed at the end of each sentence to clearly mark boundaries.
- Each token is assigned A segment embedding to indicate whether it belongs to Sentence A or Sentence B and A positional embedding to track the token's position in the input sequence.

Prediction Process:

- BERT uses the representation of the [CLS] token to classify whether the second sentence follows the first.
- The classification layer outputs a probability distribution over two labels—"is next" or "is not next"—using SoftMax.

Joint Training Objective: BERT is trained on both Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) tasks simultaneously. By minimizing the combined loss from both objectives, BERT becomes highly proficient at understanding not just individual word contexts but also the logical flow between sentences.

BERT Architecture

BERT is built on a multi-layer bidirectional Transformer encoder architecture, closely resembling the original Transformer model. While the standard Transformer consists of both an encoder and a decoder, using self-attention in the encoder and encoder-decoder attention in the decoder, BERT only uses the encoder portion. (see figure 3.2)
Architectural Differences:

- $\text{BERT}_{\text{Base}}$ includes 12 encoder layers, whereas $\text{BERT}_{\text{Large}}$ expands this to 24 layers—both significantly deeper than the 6-layer encoder described in the original Transformer paper.
- The hidden size in BERT is also larger: 768 for $\text{BERT}_{\text{Base}}$ and 1024 for $\text{BERT}_{\text{Large}}$, compared to 512 in the original Transformer.
- $\text{BERT}_{\text{Base}}$ uses 12 attention heads, and $\text{BERT}_{\text{Large}}$ uses 16, while the original Transformer architecture employs only 8.
- $\text{BERT}_{\text{Base}}$ has around 110 million parameters, while $\text{BERT}_{\text{Large}}$ contains a much larger 340 million parameters, making it one of the more powerful and resource-intensive language models developed at the time.

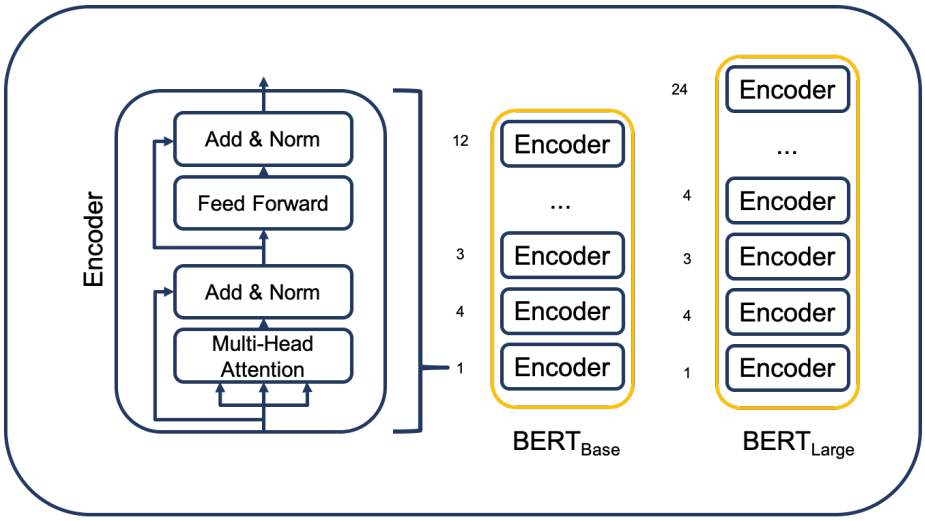


Figure 3.2: BERT Architecture

3.2 Attention Mechanism

Attention is a powerful mechanism in modern neural networks that enables models to dynamically focus on relevant parts of the input data when processing sequences. Attention mainly comes in two forms: **self-attention**, where elements within the same sequence interact with each other, and **cross-attention**, where one sequence attends to another (commonly used in encoder-decoder models). The original Transformer model introduced in the paper *"Attention Is All You Need"* by Vaswani et al. [1] also proposed **multi-head attention** as **masked multi-head attention**, which runs multiple attention operations in parallel to capture different types of contextual information simultaneously.

3.2.1 Self-Attention

Self-attention allows a model to capture relationships within an input sequence by letting each token weigh the importance of other tokens, including itself. This helps the model focus on the most relevant information.

A transformer achieves this with the help of three special matrices:

- **(Q)**uery,
- **(K)**ey, and
- **(V)**alue.

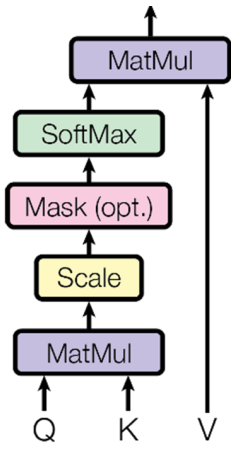


Figure 3.3: Self-Attention

The attention mechanism is computed as follows (refer figure 3.3):

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where $d_k = \frac{d_{\text{model}}}{h}$, with h being the number of attention heads, and d_{model} representing the model's hidden vector dimension.

Query and key matrices help us visualize the tokens in different embeddings. These matrices store information representing the similarity between tokens. The Q and K matrices are calculated as:

$$Q = (\text{WP Matrix})^\top_{(t \times d_{model})} \times (\text{Q-Weights Matrix})_{(d_{model} \times d_k)}$$

$$K = (\text{WP Matrix})^\top_{(t \times d_{model})} \times (\text{K-Weights Matrix})_{(d_{model} \times d_k)}$$

The WP matrix is obtained after applying the weight matrix following encoding and positional embedding. t is the context window.

Q and K weight matrices[2] act like transformation matrices that help us visualize the current coordinate system for tokens into another coordinate system that provides better separability among tokens, thereby helping the model to understand the input more effectively. As observed in figure 3.4 we are modifying the similarity in one embedding by converting it to a different embedding. In this way Q and K matrices transform the embedding to a better suited embedding for this attention problem.

While the Q and K matrices only find similarities between different features of the input, the V matrix helps in actually moving the word in question closer to those features. V is another linear transformation matrix that maps words into an embedding that is best suited for next-word prediction. (see figure 3.5)

$$V = (\text{WP Matrix})^\top_{(t \times d_{model})} \times (\text{V-Weights Matrix})_{(d_{model} \times d_k)}$$

[2]Values in Q, K and V weight matrices are between -1 and 1 and are filled randomly in the beginning. They will later be updated as the transformer starts learning.

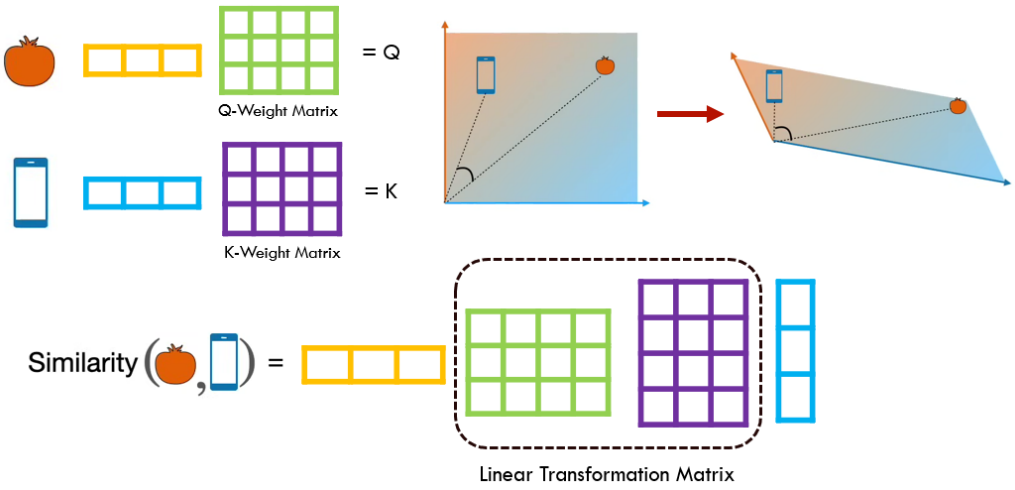


Figure 3.4: Linear Transformation Matrix derived from Q and K

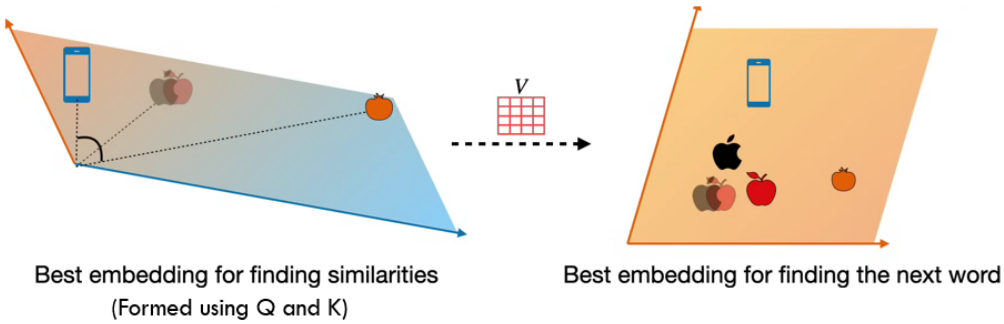


Figure 3.5: Linear Transformation by V

Table 3.2: Comparison of Q, K, and V matrices

Q and K Matrices	V Matrix
Coordinates of words are not changed in the embedding	Coordinates for words are changed in the embedding
Does not modify original WP matrix	It modifies the original WP matrix
Used to find best similarity	Used to find best next word in the prediction

Cross-Attention

While self-attention focuses on capturing dependencies within the same sequence, **cross-attention** extends this concept by allowing the model to attend to a different input sequence. This is particularly useful in encoder-decoder architectures, such as in machine translation tasks, where the decoder uses cross-attention to focus on relevant parts of the encoder's output. Just like self-attention, cross-attention also relies on the Q, K, and V matrices, but with a key difference: the Q is derived from the decoder input, while K and V are computed from the encoder output. This enables the decoder to selectively extract information from the encoded source sequence to guide the generation of the output sequence.

$$
\begin{aligned}
Q &= (\text{Decoder Output})^T \times \text{Q-Weights Matrix} && (\text{Shape: } T_d \times d_k) \\
K &= (\text{Encoder Output})^T \times \text{K-Weights Matrix} && (\text{Shape: } T_e \times d_k) \\
V &= (\text{Encoder Output})^T \times \text{V-Weights Matrix} && (\text{Shape: } T_e \times d_v)
\end{aligned}
$$

Here, T_d is the number of tokens in the decoder input, T_e is the number of tokens in the encoder output, and d_k, d_v are the dimensions of the key and value vectors respectively. This formulation allows the decoder to attend over encoder representations and generate contextually rich outputs.

3.2.2 Multi-head Attention

The mechanism that enables the Transformer to achieve parallelism and capture diverse aspects of the input representation is known as **Multi-Head Attention**. Instead of applying a single self-attention function, the Transformer uses multiple attention heads—each with its own set of learnable weight matrices W_Q^i, W_K^i, and W_V^i. These heads act like independent *workers*, each focusing on different subspaces or relationships within the sequence. Here multiple single-head blocks work in parallel as shown in figure 3.6.

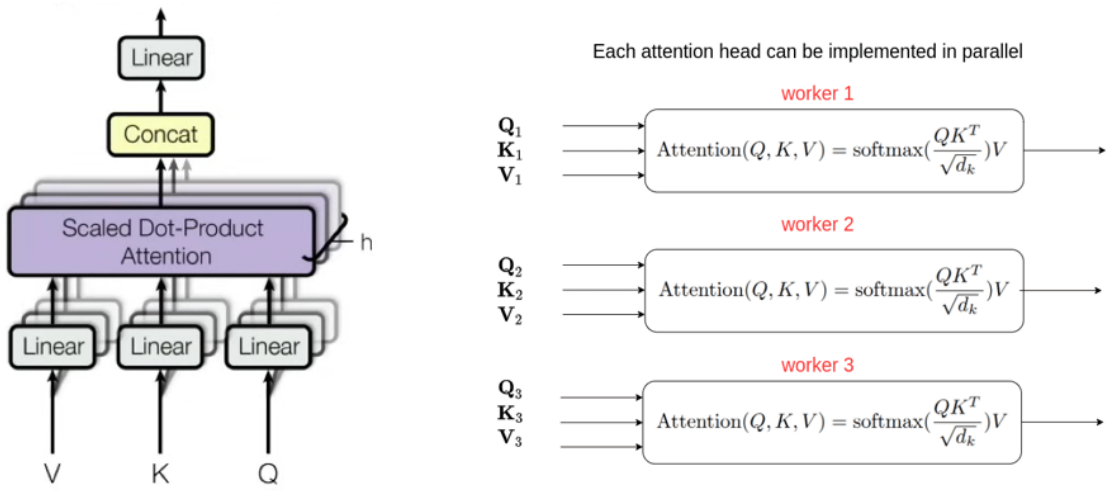


Figure 3.6: Multi-Head Attention

Mathematically, the output of multi-head attention is given by:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head is defined as:

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Here:

- Q, K, V are the input query, key, and value matrices.
- W_Q^i, W_K^i, and W_V^i are the learnable projection matrices for the i-th head.
- W^O is the output projection matrix applied after concatenation.
- h is the number of attention heads.

The concatenated output has a dimensionality of $d_k \times (N \cdot h)$, where N is the length of the input sequence and d_k is the dimensionality of each head. Since the original input to the multi-head attention layer has dimensionality $d_k \times$ (number of tokens), an additional linear transformation (via W^O) is applied to map the concatenated output back to the expected shape. This is accomplished by applying a linear transformation using the matrix W^O. The values in this matrix are initialized randomly, typically with values between 0 and 1, and

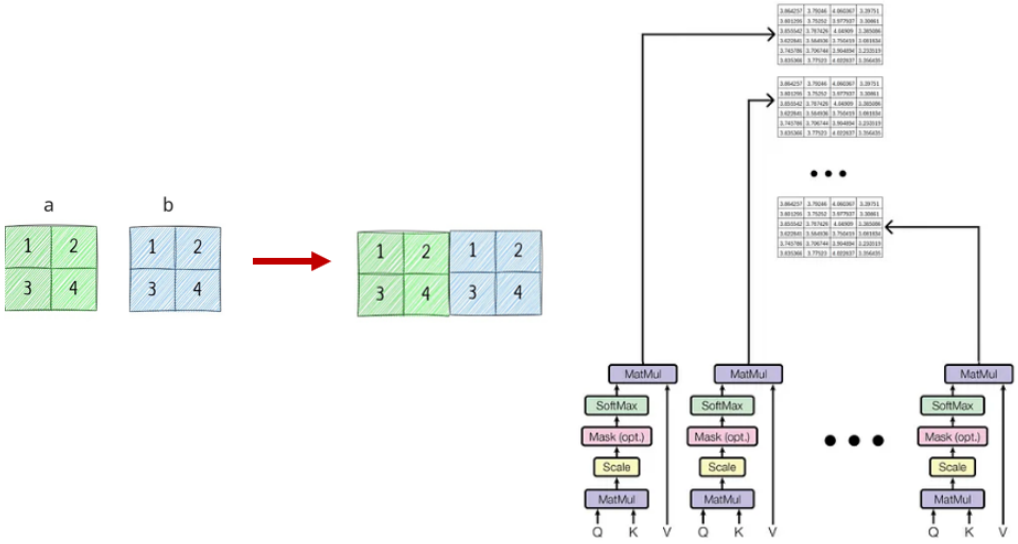


Figure 3.7: Multi-head concatenation

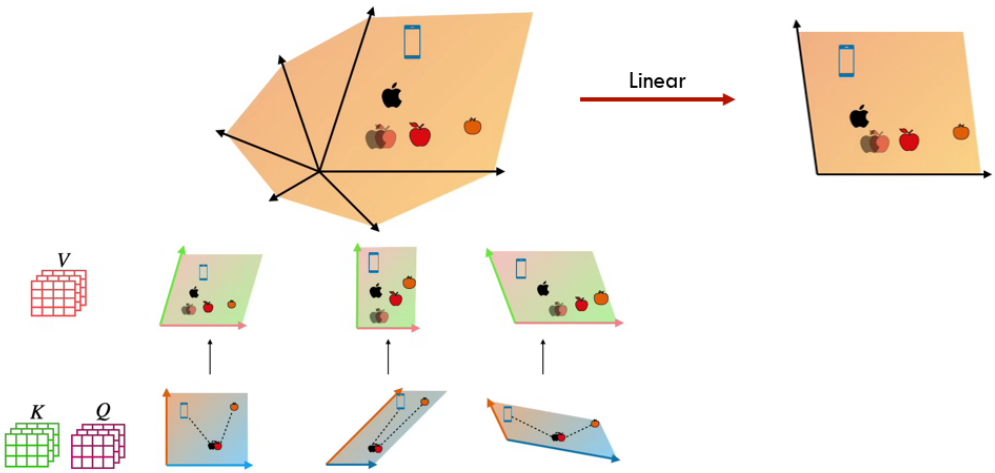


Figure 3.8: Multi-head Attention output

are updated during the training process as the Transformer learns. Concatenation happens along the x-axis as shown in figure 3.7.

This design allows the model to jointly attend to information from different representation subspaces at different positions. Figure 3.8 shows each attention head computes attention in parallel, and their outputs are then concatenated and projected using a final linear layer to obtain the final representation.

Masked Multi-head Attention

This is the first attention layer in decoder layer. This is nothing but a multi-head attention but with masking of some values. This attention is a self-attention where attention score between decoder tokens are calculated and establish relationship between them. While predicting one token is fed to decoder at a time and hence to mask the future possible values being predicted before only the tokens next to present token are masked. This way model doesn't predicts next tokens before only and learn token by token of the target sentence. Table 3.3 summarizes the attention layers used in encoder and decoder.

Table 3.3: Comparison of Multi-head Attention Types in Transformer Architecture

Parameters	**Multi-head Attention in Encoder**	**Masked Multi-head Attention in Decoder**	**Multi-head Attention in Decoder (Cross Attention)**
Location	Encoder block	Decoder block	Decoder block
Attention Type	Self attention	Self attention	Cross attention
Inputs	Source tokens	Target tokens	Source–target token pairs
Outputs	Attention between source tokens	Attention between target tokens	Attention between source and target tokens
Masking	Absent	Present	Absent
Q, K, V	Q, K, V come from the output of previous encoder block	Q, K, V come from the output of previous decoder block	Q comes from previous decoder layer, K and V come from the encoder output
Formula	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T+M}{\sqrt{d_k}}\right)V$	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

3.3 Need for Pruning

3.3.1 Redundancy in Attention layers

Michel et al. [11] in his paper "Are Sixteen Heads Really Better than One?" made a surprising observation that even if models have been trained using multiple heads, in practice, a large percentage of attention heads can be removed without significantly impacting performance. In fact, some layers can even be reduced to a single head. We verified this fact by examining heat maps of the attention heads as shown in figure 3.9. For this we built a

basic custom transformer as given in original paper by Vaswani et al [1]. This transformer has 6 encoder-decoder layers and 8 attention heads. We used English sentences from "English to Marathi machine translation" dataset from Kaggle. After training using a Word level tokenizer, visualization was performed using the source text *"However, malls, market complexes and big shops will remain closed."*. It was found that layer 5 head 1 contains less useful information and can be pruned without loss of accuracy. Similarly other heads were also found to be redundant and were candidates for pruning.

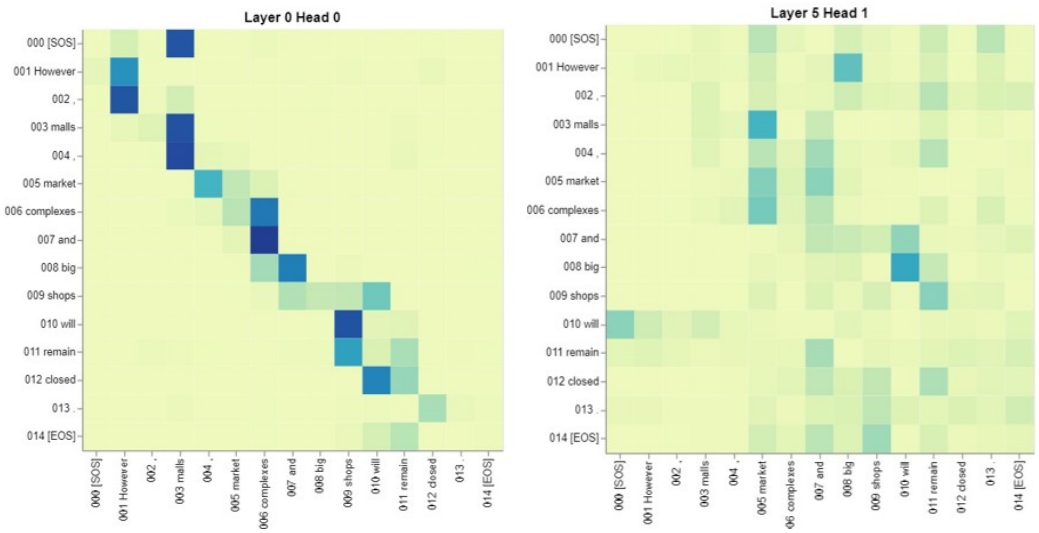


Figure 3.9: Heat maps of attention heads extracted from the transformer

3.3.2 Redundancy in FFN layers

Just like attention heads in transformer models often exhibit redundancy—where multiple heads learn similar patterns or contribute marginally to the model's output—the FFN layers also contain redundant neurons or channels. In many cases, different neurons in the FFN layers learn overlapping representations or remain inactive for most inputs, leading to inefficiencies in computation and memory usage. Studies have shown that a significant fraction of FFN activations contribute minimally to the final output, suggesting that these layers are over-parameterized. This redundancy forms a strong motivation for pruning, which can reduce model size and inference cost without significantly affecting performance. Pruning FFN layers, like attention heads, helps streamline the model, enabling deployment in resource-constrained environments while preserving accuracy.

Chapter 4

METHODOLOGY

4.1 Proposed Work

To evaluate the impact of pruning on transformer models, we employ a systematic approach:

1. **Head Importance Analysis in Attention Layers:** We use an importance score and analyze the contribution of each attention head using gradient-based and information-theoretic methods.

2. **Neuron Importance Analysis in FFN Layers:** We use an importance score to analyze which neurons are redundant and need to be pruned using saliency-based, structured pruning methods.

3. **Dual-step Pruning:** We employ a pruning strategy that jointly optimizes attention heads and FFN neurons to achieve a better trade-off between efficiency and accuracy. Figure 4.1 summarizes an overall view of steps 1 to 3.

4. **Evaluation of Multiple Transformer models and Tasks:** We conduct experiments on various Transformers and NLP benchmark tasks, comparing pruned models against baseline transformers. We use pre-trained transformers from the Hugging Face Library [12] which have state-of-the-art Transformer architectures under a unified API. Initial experimentation prunes the `bert-base-uncased` model which is 12-layer, 768-hidden, 12-heads, 110M parameter model, trained on lower-cased English text. We then extend the use of this algorithm on different transformers against different GLUE tasks. After training, to perform comparisons, the baseline and pruned

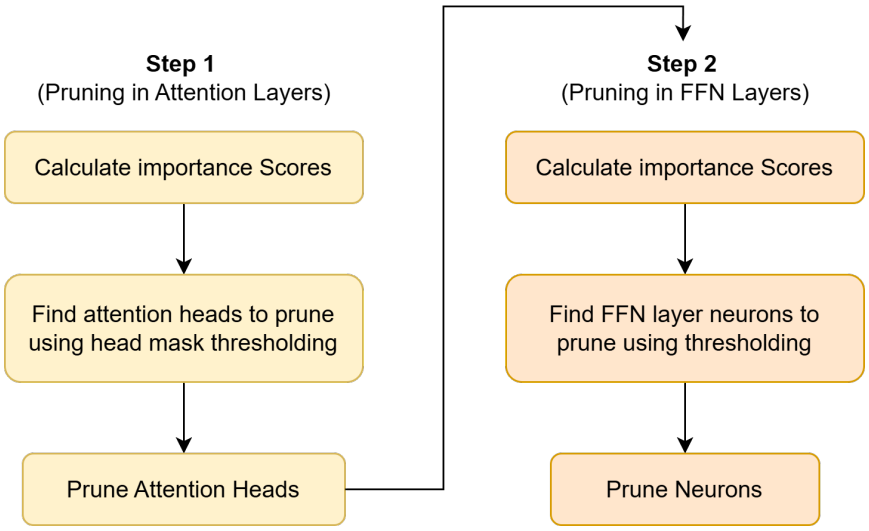

Step 1
(Pruning in Attention Layers)
Calculate importance Scores
Find attention heads to prune using head mask thresholding
Prune Attention Heads
Step 2
(Pruning in FFN Layers)
Calculate importance Scores
Find FFN layer neurons to prune using thresholding
Prune Neurons

Figure 4.1: Flowchart of steps used in the dual-step pruning algorithm.

transformer undergo a benchmark to find accuracy scores, inference time, memory usage and total no of parameters. For this we use `PyTorchBenchmark` from hugging face library which allows to flexibly benchmark the Transformers models. Then we benchmark the pruned `bert-base-uncased` model on different GLUE tasks to compare the accuracy of the baseline model against the pruned model.

5. **Inference on a Question-Answering Task:** We fine-tune the baseline and the pruned `bert-base-uncased` model on Stanford Question Answering Dataset (SQuAD) [13] which is a reading comprehension dataset built from Wikipedia articles, where crowdworkers generate questions. Each answer is either a text span directly taken from the related passage or, in some cases, no answer is possible at all. We then use a custom context and ask a question to the model. Further, we assess and compare the outputs of the fine-tuned baseline model and the fine-tuned pruned model.

6. **Analysis and Comparison:** We compare the impact of pruning transformer models by evaluating major performance indicators such as accuracy, inference time, memory usage, and total number of parameters between the baseline and pruned models. On the basis of this comparative study, we obtain insights and make conclusions about the trade-offs and advantages brought in by the pruning process. We then answer the major questions our research sought to answer.

Name	Download	More Info	Metric
The Corpus of Linguistic Acceptability			Matthew's Corr
The Stanford Sentiment Treebank			Accuracy
Microsoft Research Paraphrase Corpus			F1 / Accuracy
Semantic Textual Similarity Benchmark			Pearson-Spearman Corr
Quora Question Pairs			F1 / Accuracy
MultiNLI Matched			Accuracy
MultiNLI Mismatched			Accuracy
Question NLI			Accuracy
Recognizing Textual Entailment			Accuracy
Winograd NLI			Accuracy
Diagnostics Main			Matthew's Corr

Figure 4.2: A list of tasks in GLUE benchmark

4.2 GLUE Benchmark

The General Language Understanding Evaluation (GLUE) benchmark is a large collection of diverse natural language understanding tasks designed to assess the generalization, robustness, and linguistic capabilities of machine learning models, in particular transformer-based models. [14] It encompasses a broad variety of tasks like sentiment analysis (SST-2), linguistic acceptability (CoLA), natural language inference (MNLI, RTE, and QNLI), paraphrase identification (MRPC and QQP), and so on, providing a highly balanced evaluation framework. The following figure 4.2 shows all the tasks that GLUE supports:

Due to its broad coverage of tasks and popularity throughout the NLP community, GLUE is used as a standard measure for the comparison of model performance. For this project, we utilize the GLUE benchmark to evaluate the effectiveness of our pruning techniques against transformer models. This evaluation will serve to attest to the actual-world practicality of our pruning framework when efficiency and scalability are critical. We use Semantic Textual Similarity Benchmark (SST-2) for the first experimentation and then generalize to other tasks of the list.

4.3 Importance Scores

We calculate importance score to find out the contribution of each attention head or FFN neurons. For attention heads we use a weighted sum of the normalized gradients and normalized entropy.

$$importance\_score_{head} = \alpha \times normalized\_gradients + (1-\alpha) \times (1-normalized\_entropy)$$

Gradients measure how much changing the head's output would affect the loss. Higher gradient norms generally indicate the head is more significant since its output has a bigger influence on the loss. Entropy indicates how sharp or confident the head's attention is. Lower entropy (more peaked attention) typically indicates the head is more confident and more useful. α lets us balance between gradient-based (sensitivity) and entropy-based (confidence) signals. Hence, the formula suggests that the attention heads with a low score are potential candidates for pruning.

Similarly, for FFN neurons we use an absolute product of the normalized activations and the normalized gradients.

$$importance\_score_{neuron} = |normalized\_activations \times normalized\_gradients|$$

Activations tell us how active a neuron is throughout the data. If a neuron never activates (always close to zero), it's probably useless. Gradients tell us how much effect changing the neuron's output has on the loss. High gradient means that neuron's output is important. By taking the product of the two, we want to identify neurons that are both active and influential. Therefore, the formula implies that the neurons with a low score are good candidates for pruning.

4.4 Algorithm

We use an iterative head masking and pruning algorithm for Transformer models to reduce computational complexity while controlling performance degradation. The method begins by initializing a binary head mask $H \in 0, 1^{L \times N}$, where L is the number of layers and N is the number of attention heads per layer. Two accumulators are initialized: the head

importance matrix I and the head entropy matrix E, both set to zero. For each mini-batch sampled from the evaluation dataset $\mathcal{D}$, model outputs are computed with the current head mask, and gradients of the loss $\mathcal{L}$ with respect to H are obtained via backpropagation. The importance matrix I is updated by accumulating the absolute gradient values, while the entropy E is updated by calculating the entropy of each attention head's distribution.

Once the entire dataset is processed, I and E are normalized by the total number of tokens. In the masking phase, each head is assigned a combined score S, defined as a weighted combination of normalized importance $\tilde{I}$ and one minus normalized entropy $\tilde{E}$, where the hyperparameter $\alpha \in [0, 1]$ controls the trade-off between relying more on importance (high α) versus entropy (low α). Heads are ranked based on S, and the bottom μ_h fraction (where $\mu_h \in (0, 1)$) of the currently active heads are masked in each iteration. After masking, the model's evaluation metric $M(H)$ is recomputed on $\mathcal{D}$, and the process repeats until the metric drops below a threshold τM_0, where M_0 is the original unpruned model's score and $\tau \in (0, 1)$ determines the acceptable performance degradation.

Finally, all masked heads are permanently pruned from the model using its native pruning functionality. This method is summarized in Algorithm 1.

Algorithm 1 Head Masking and Pruning for Transformer Models

Require: Evaluation dataset $\mathcal{D}$, pretrained model with L layers and N attention heads, metric function M, original metric score M_0, threshold factor $\tau \in (0, 1)$, masking fraction $\mu_h \in (0, 1)$, weighting factor $\alpha \in [0, 1]$

1: **Initialize:** Set head mask

$$H \leftarrow \mathbf{1}_{L \times N}$$

2: Initialize accumulators for importance $I \in R^{L \times N}$ and entropy $E \in R^{L \times N}$ to zeros.

3: **for** each mini-batch (x, y) from $\mathcal{D}$ **do**

4: Compute model outputs with current head mask H:

$$\text{outputs} = \text{Model}(x; H)$$

5: Compute loss $\mathcal{L}$ and perform backward propagation to obtain gradients w.r.t. H.

6: **Accumulate head importance:**

$$I \leftarrow I + |\nabla_H \mathcal{L}|$$

7: **Compute attention entropy:** For each layer ℓ,

$$E_\ell = -\sum_{i=1}^{N} A_{\ell,i} \log(A_{\ell,i})$$

where $A_{\ell,i}$ is the attention distribution for head i in layer ℓ.

8: **end for**

9: Normalize accumulators by the total number of tokens:

$$I \leftarrow \frac{I}{T}, \quad E \leftarrow \frac{E}{T}$$

10: **Iterative Masking:** Let $M(H)$ denote the evaluation metric computed using head mask H.

11: **while** $M(H) \geq \tau\, M_0$ **and** there exist unmasked heads in H **do**

12: Compute the combined score for each head:

$$S = \alpha\, \tilde{I} + (1 - \alpha) \left(1 - \tilde{E}\right)$$

13: **Rank** all heads (flattened over layers) in ascending order of S.

14: Select the bottom $\mu_h \cdot (L \times N)$ heads that are not yet masked:

$$\mathcal{H}_{\text{mask}} \leftarrow \text{ArgSort}(S)[1 : \lceil \mu_h \cdot (L \cdot N) \rceil]$$

15: Update the head mask:

$$H_{(i,j)} \leftarrow 0 \quad \forall (i,j) \in \mathcal{H}_{\text{mask}}$$

16: Recompute the metric with the updated H:

$$M(H) \leftarrow \text{Evaluate}(H, \mathcal{D})$$

17: **end while**

18: **Pruning:** For each layer $\ell = 1, \ldots, L$, define

$$\mathcal{P}_\ell = \{i \in \{1, \ldots, N\} : H_{\ell,i} = 0\}.$$

Then prune heads in $\mathcal{P}_\ell$ via the model's built-in pruning function.

Ensure: Pruned model with reduced parameter count $M(H)$.

Hyperparameters α, μ_h, and τ respectively is alpha that control the importance-entropy trade-off, masking fraction that control masking aggressiveness per iteration, and the masking threshold which is maximum allowed accuracy drop.

After attention head pruning, we apply neuron-level pruning to the feed-forward (FFN) modules of the Transformer model to further reduce its size and computational cost. First, for each linear FFN layer m, we initialize a neuron mask $mask_m$ set to ones. The model's original performance M_0 is evaluated on the dataset $\mathcal{D}$. Then, for each mini-batch, a forward pass is performed using the current neuron masks, followed by backward propagation to compute the loss gradients. For each FFN module, the cached activations and gradients are used to estimate neuron importance: specifically, by averaging the absolute element-wise product of activations and gradients across all tokens and batches. This provides an importance score $I_m \in R^{d_o}$ for each output neuron.

After processing all batches, the importance scores are averaged, and a pruning threshold τ_m is determined using the μ_n-th quantile, where $\mu_n \in (0, 1)$ is a hyperparameter that controls the fraction of neurons to retain (higher μ_n keeps more neurons). Neurons with importance scores below τ_m are masked out by zeroing the corresponding rows in the weight matrix and, if present, the bias vector. Finally, the pruned model M is re-evaluated to obtain the new metric M_{pruned}, measuring any performance loss. This method is summarized in Algorithm 2.

Algorithm 2 Neuron Pruning in Transformer FFN Layers

Require: Model M (after head pruning), dataset $\mathcal{D}$, pruning quantile $\mu_n \in (0, 1)$

Initialize: For each FFN module m, set mask $\textbf{mask}_m \leftarrow \mathbf{1}$

Compute original metric:

$$M_0 \leftarrow \text{Evaluate}(M, \mathcal{D})$$

for each batch $(x, y_{\text{true}}) \in \mathcal{D}$ **do**

  Forward pass: $y = M(x; \{\textbf{mask}_m\})$

  Compute loss $\mathcal{L}(y, y_{\text{true}})$ and perform backward pass

  for each FFN module m **do**

    Update neuron importance:

$$I_m \leftarrow I_m + |A_m \odot G_m| \quad \text{(sum over batch and tokens)}$$

  end for

end for

Normalize importance scores I_m over total number of samples

for each module m **do**

  Set threshold $\tau_m = \text{Quantile}(I_m, \mu_n)$

  Mask neurons:

$$\widetilde{I}_m[i] = \begin{cases} 1, & I_m[i] \geq \tau_m \\ 0, & \text{otherwise} \end{cases}$$

  Prune weights and biases:

$$W_m[i, :] \leftarrow W_m[i, :] \cdot \widetilde{I}_m[i], \quad b_m[i] \leftarrow b_m[i] \cdot \widetilde{I}_m[i]$$

end for

Re-evaluate:

$$M_{\text{pruned}} \leftarrow \text{Evaluate}(M, \mathcal{D})$$

Ensure: Pruned model M and metric M_{pruned}

The hyperparameter μ_n is the neuron quantile threshold which controls the sparsity of the FFN layers: smaller μ_n results in more aggressive pruning and higher compression, at the risk of greater performance degradation.

Figure 4.3 shows an implementation workflow of the above algorithms for the BERT model. The path in green is for the baseline model. The blue path is for pruned model.

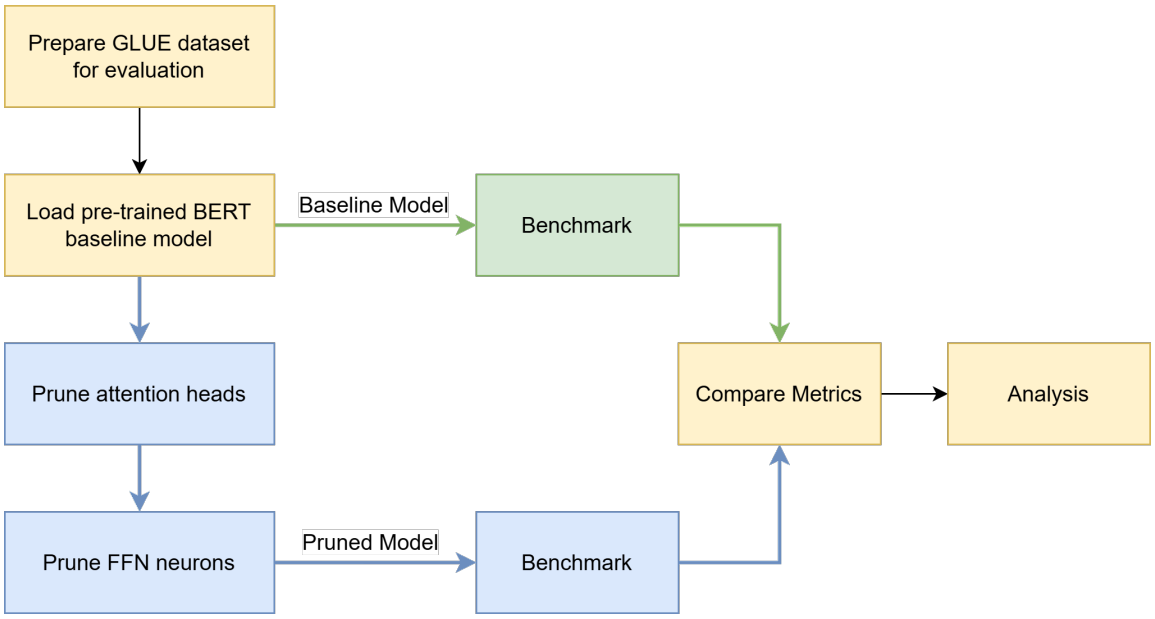


Figure 4.3: Algorithm Implementation Workflow for BERT model

Chapter 5

IMPLEMENTATION

5.1 Setup

Softwares: Windows 11 x64, Visual Studio Code, Kaggle Notebooks (Online)

Languages: Python 3.9, Batch Script

Libraries: PyTorch, Numpy, Transformers (HuggingFace), Datasets (HuggingFace), TensorBoard, Pandas

Hardware:

Local: AMD Ryzen 7 7840HS @3.8GHz, 16GB RAM, NVIDIA RTX 4050 6GB GPU

Kaggle: 30GB RAM, 2x NVIDIA Tesla T4 15GB GPUs, NVIDIA P100 16GB GPU

5.2 Experiments

Implementation consisted of a series of 4 experiments. See figure 5.2 and figure 5.3 for detailed flowcharts on the entire project implementation.

1. Experiment 1 began with the standard `bert-base-uncased` model. The dual-step pruning algorithm was applied to this model using the SST-2 task from the GLUE benchmark. The resulting output was a pruned BERT model whose metrics such as GLUE score, total number of parameters, etc. were compared. This experiment was conducted on a local workstation.

2. Experiment 2 extended the analysis by applying the same pruning algorithm to several transformer models from the BERT family, including `bert-large-uncased`,

`roberta-base`, and `distilbert-base-uncased`, all evaluated on the SST-2 task. In this case, inference time and memory usage across models were compared. This experiment was also executed on the local workstation.

3. Experiment 3 involved fine-tuning the BERT model on multiple GLUE tasks, specifically SST-2, MRPC, RTE, and WNLI—after pruning. Given the higher memory requirements, this experiment was run on Kaggle Notebooks.

4. Finally, in Experiment 4, the pruned model was fine-tuned for a real-world application: a question answering task using the SQuAD dataset. Inference was then performed on two different contexts to evaluate performance. This final experiment was also carried out on Kaggle Notebooks.

5.3 Performance Benchmarking Tools

HuggingFace Transformer library provides a handy tool to measure inference time and memory usage by utilizing python inbuilt functions and NVIDIA libraries. The Colab Notebook [15] provides comprehensive guide on how to benchmark transformer models.

The code is based on `PyTorchBenchmark` framework. The code measures inference time as follows: First, it prepares a callable function that performs a forward pass on the model with a batch of randomly generated input data (using the model's configuration and the appropriate input size). It uses Python's `timeit` library to execute the function several times (typically 10 iterations per run, repeated a number of times) and takes the minimum of these runtimes divided by the number of iterations to get an average per-call inference time. In short, inference time is determined by running the model's forward pass multiple times on test data, timing these runs, and then normalizing the total time to get an average time per inference call.

Similarly more memory calculations, the code uses NVIDIA's `NVML` tools to check the GPU's current memory usage right after the function runs. It initializes NVML, runs the model's forward (or training) pass, and then queries the GPU to see how many bytes are in use, wrapping this measurement in a Memory object that the rest of the benchmark framework can understand.

```
====================       INFERENCE - SPEED - RESULT       ================
--------------------------------------------------------------------------------
Model Name             Batch Size     Seq Length     Time in s
--------------------------------------------------------------------------------
bert-base-uncased          8               8             0.006
bert-base-uncased          8               32            0.006
bert-base-uncased          8               128           0.018
bert-base-uncased          8               512           0.088
--------------------------------------------------------------------------------

====================      INFERENCE - MEMORY - RESULT       ================
--------------------------------------------------------------------------------
Model Name             Batch Size     Seq Length    Memory in MB
--------------------------------------------------------------------------------
bert-base-uncased          8               8             1227
bert-base-uncased          8               32            1281
bert-base-uncased          8               128           1307
bert-base-uncased          8               512           1539
--------------------------------------------------------------------------------
```

Figure 5.1: Output of HuggingFace Transformer Benchmark Tool

One can simply instantiate an object as follows

```
args = PyTorchBenchmarkArguments(
models=["bert-base-uncased"],
batch_sizes=[8],
sequence_lengths=[8, 32, 128, 512]
)
benchmark = PyTorchBenchmark(args)
```

An instantiated benchmark object can then be run by calling `benchmark.run()`. Figure 5.1 shows output of the benchmarking tool for `bert-based-uncased` model with `batch size = 8` and `seq length = [8,32,128,512]`. This benchmarking tool was used in Experiment 2.

HuggingFace Transformer library provides another script [16] designed to fine-tune pre-trained transformer models for sequence classification tasks specifically for GLUE benchmarks. It begins by parsing command-line arguments using dataclasses, which allow users to specify model paths, dataset configurations, and training parameters. It then

checks for library version requirements and sets up logging and environment variables. The code downloads or loads a dataset (either from the HuggingFace Hub or local files) based on the provided task or dataset name, and determines label formatting and configuration details by inspecting dataset features. The tokenizer is used to preprocess the dataset, truncating and padding sequences according to the specified maximum sequence length, and a mapping function transforms raw examples into model inputs.

Next, the script creates a Trainer instance from the Transformers library, which encapsulates training, evaluation, and prediction logic, handling metrics computation, checkpoint recovery, and distributed training. Finally, it executes training, conditionally performs evaluation and prediction steps, and handles saving metrics and model outputs. It even supports pushing to the HuggingFace Hub or generating a model card as part of the workflow. This script was used in Experiment 3.

The fine-tuning of `bert-base-uncased` on the MRPC Task can be done using the following command:

```
python run_glue.py \
  --model_name_or_path "bert-base-uncased" \
  --task_name mrpc \
  --do_train \
  --max_seq_length 128 \
  --per_device_train_batch_size 32 \
  --learning_rate 2e-5 \
  --num_train_epochs 5 \
  --output_dir mrpc
```

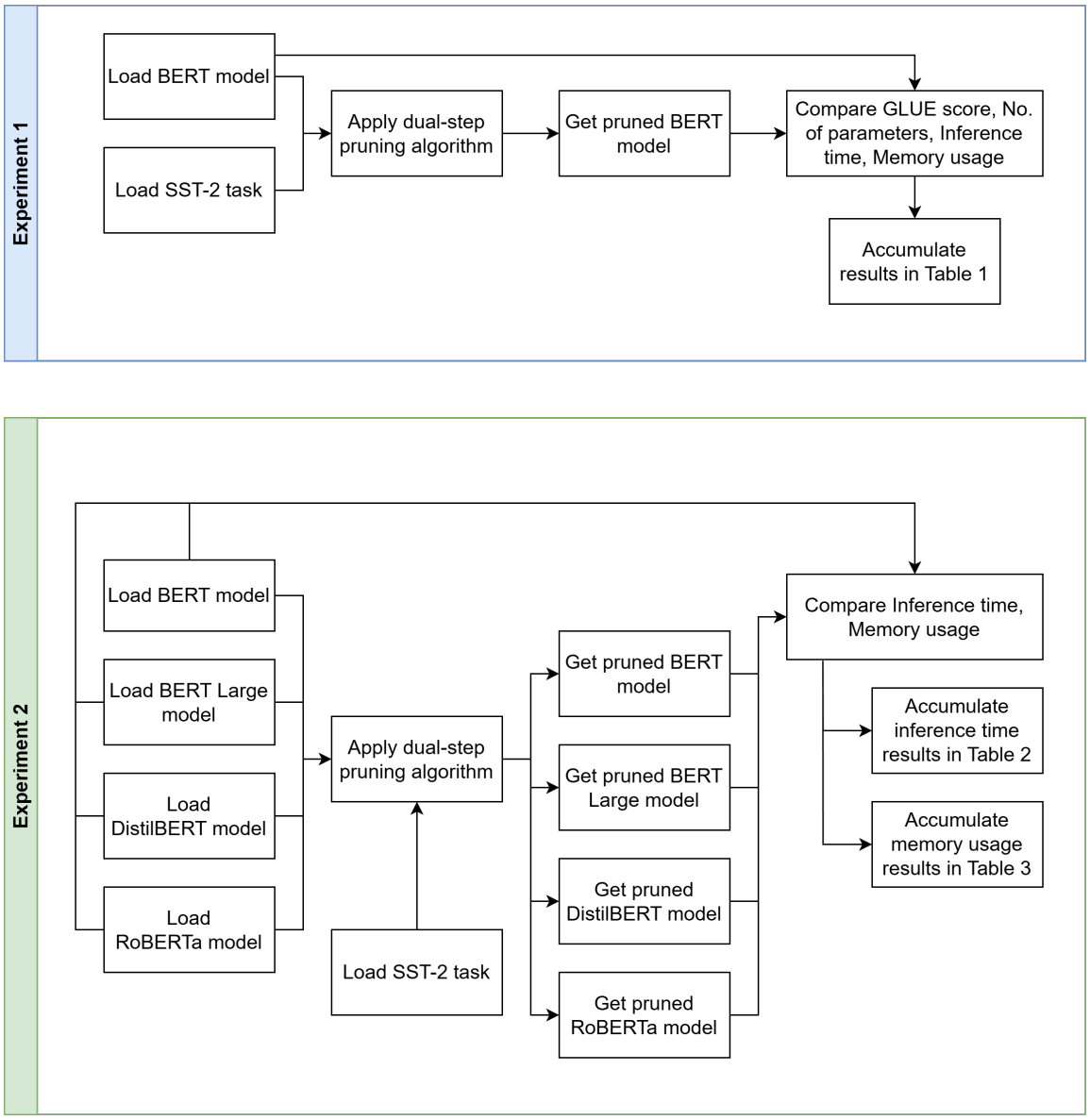

Experiment 1
Load BERT model
Load SST-2 task
Apply dual-step pruning algorithm
Get pruned BERT model
Compare GLUE score, No. of parameters, Inference time, Memory usage
Accumulate results in Table 1
Experiment 2
Load BERT model
Load BERT Large model
Load DistilBERT model
Load RoBERTa model
Apply dual-step pruning algorithm
Load SST-2 task
Get pruned BERT model
Get pruned BERT Large model
Get pruned DistilBERT model
Get pruned RoBERTa model
Compare Inference time, Memory usage
Accumulate inference time results in Table 2
Accumulate memory usage results in Table 3

Figure 5.2: Implementation Workflow of Experiment 1 and 2

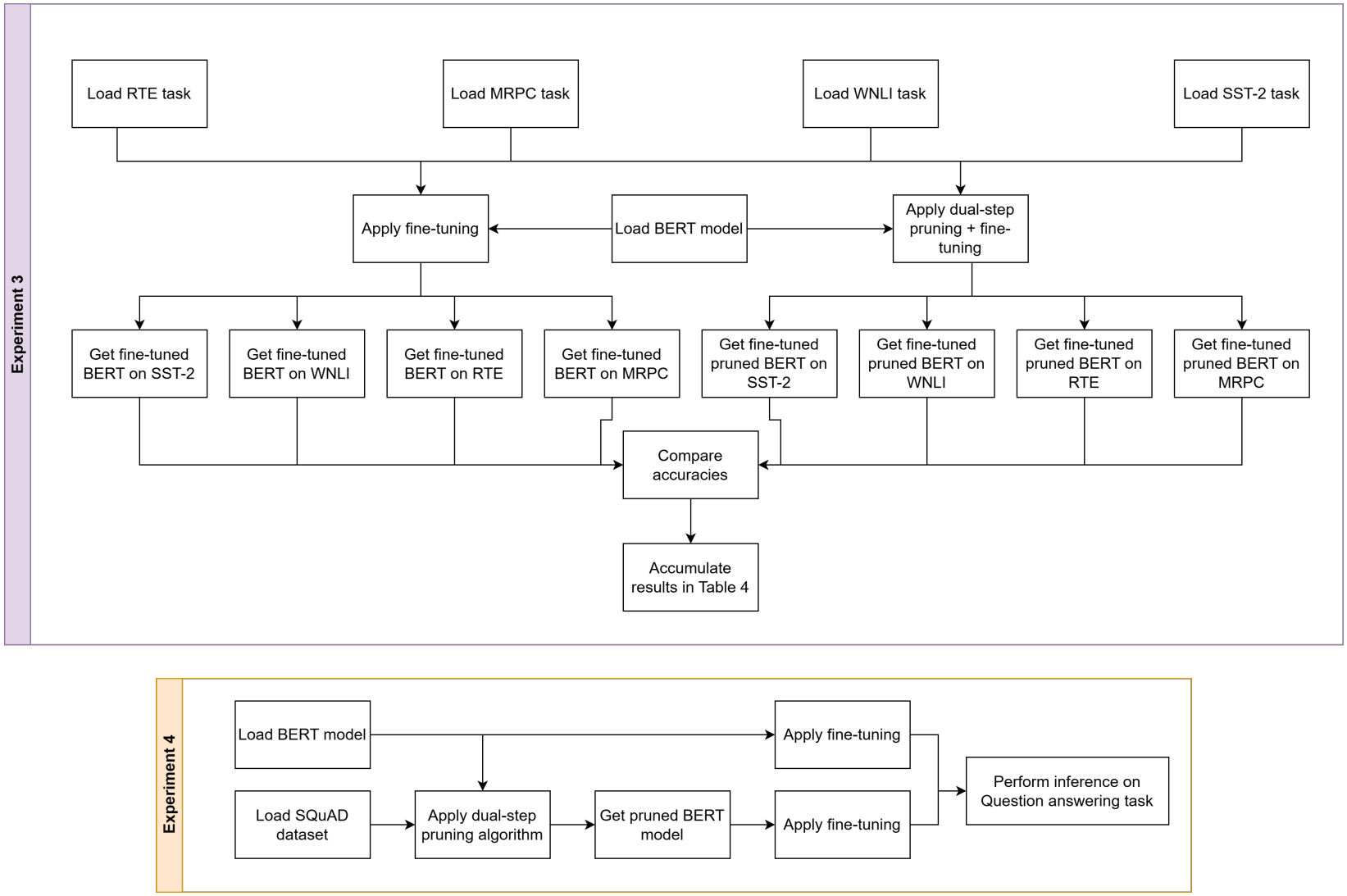


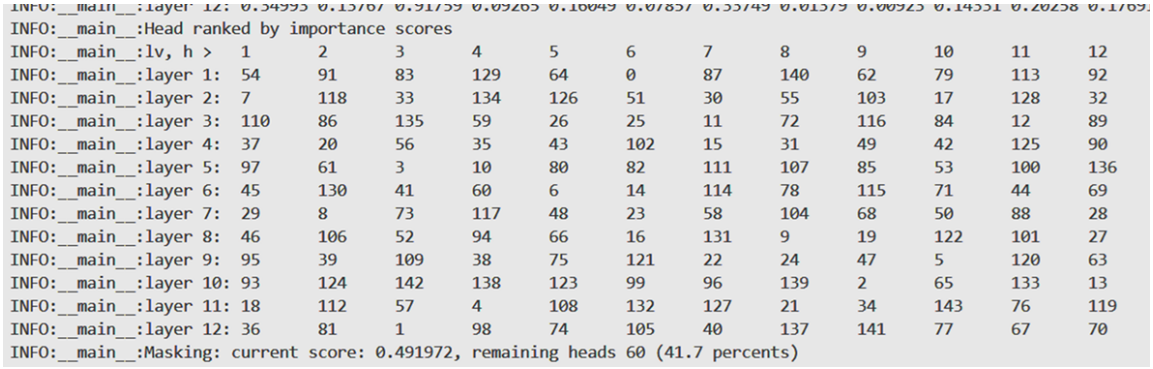
Figure 5.3: Implementation Workflow of Experiment 3 and 4

Chapter 6

RESULTS AND OBSERVATIONS

6.1 Metric Comparison

To evaluate the performance of multiple transformers on the algorithm, we conducted rigorous experiments. Running the algorithm on a transformer yield a table displaying the attention heads ranked by their importance scores. The rows indicate the layers while the columns indicate the attention heads. These values are visualized in figure 6.1.



```
INFO:__main__:Head ranked by importance scores
INFO:__main__:lv, h >   1     2     3     4     5     6     7     8     9     10    11    12
INFO:__main__:layer 1:  54    91    83    129   64    0     87    140   62    79    113   92
INFO:__main__:layer 2:  7     118   33    134   126   51    30    55    103   17    128   32
INFO:__main__:layer 3:  110   86    135   59    26    25    11    72    116   84    12    89
INFO:__main__:layer 4:  37    20    56    35    43    102   15    31    49    42    125   90
INFO:__main__:layer 5:  97    61    3     10    80    82    111   107   85    53    100   136
INFO:__main__:layer 6:  45    130   41    60    6     14    114   78    115   71    44    69
INFO:__main__:layer 7:  29    8     73    117   48    23    58    104   68    50    88    28
INFO:__main__:layer 8:  46    106   52    94    66    16    131   9     19    122   101   27
INFO:__main__:layer 9:  95    39    109   38    75    121   22    24    47    5     120   63
INFO:__main__:layer 10: 93    124   142   138   123   99    96    139   2     65    133   13
INFO:__main__:layer 11: 18    112   57    4     108   132   127   21    34    143   76    119
INFO:__main__:layer 12: 36    81    1     98    74    105   40    137   141   77    67    70
INFO:__main__:Masking: current score: 0.491972, remaining heads 60 (41.7 percents)
```

Figure 6.1: Heads ranked by importance scores

After computing the importance scores, the algorithm generates a dictionary of key values pairs that tell which attention head need to be pruned from which layer. The following example shows a sample output of the algorithm.

```
Heads to prune:
{0: [0, 1, 4, 5, 6, 7, 8, 9, 11],
1: [0, 2, 3, 5, 7, 8, 9, 10, 11],
2: [3, 4, 5, 6, 7, 8, 10, 11],
3: [1, 2, 3, 4, 6, 7, 8, 10, 11],
```

```
4: [0, 1, 4, 6, 8, 9, 10],
5: [2, 4, 8],
6: [7],
7: [0, 5, 8],
8: [1, 3, 7, 8, 9, 11],
9: [2, 5, 8, 10, 11],
10: [0, 3, 7, 8, 11],
11: [2, 5, 6, 8, 9]}
```

This says that head 0,1,4,5,6,7,8,9,11 need to be pruned from layer 0 and so on.

Table 6.1 compares the glue score, total no of parameters, inference time and the memory usage of the baseline and pruned `bert-base-uncased` model on SST-2 task. For this experiment we used `masking threshold = 0.92` and `alpha = 0.5` giving equal importance to entropy and gradients and a `neuron fraction = 0.1`. The model was trained on a single NVIDIA RTX 4050 6GB GPU with a `batch size = 4` and `max sequence length = 128`. The pruned model saw a massive 30.14% decrease in inference time and a 14.01% reduction in memory usage with only a 3.64% drop in the GLUE score.

Table 6.1: Comparing Baseline vs. Pruned `bert-base-uncased` model on SST-2

Metric	**Baseline**	**Pruned**	**% Change**
GLUE Score	0.5355	0.5160	-3.64%
Total no. of Parameters	109483778	92952578	-15.09%
Inference Time (s)	0.0647	0.0452	-30.14%
Memory Usage (MB)	771	663	-14.01%

Table 6.2 and Table 6.3 compares inference time in seconds and memory usage in MB respectively for different transformer models evaluated on SST-2 task. Same configuration of hyperparameter was used in this experiment compared to the previous experiment with only change in the transformer model. `roberta-base` and `distilbert-base-uncased` showed little to no drop in GLUE score while showing a huge improvement in inference. The `distilbert-base-uncased` model showed only a 4.06% drop in memory usage indicating how efficiently the model is already optimized that further reduction is negligible.

Table 6.2: Inference Time Comparison of Baseline vs. Pruned model on SST-2

Transformer model	GLUE	Inference Time (s)		
	% Change	Baseline	Pruned	% Change
bert-base-uncased	-3.64%	0.0647	0.0452	-30.14%
bert-large-uncased	-4.41%	0.1950	0.1150	-41.02%
roberta-base	-0.01%	0.0712	0.0393	-44.80%
distilbert-base-uncased	0.00%	0.0734	0.0629	-14.31%

Table 6.3: Memory Usage Comparison of Baseline vs. Pruned model on SST-2

Transformer model	GLUE	Memory Usage (MB)		
	% Change	Baseline	Pruned	% Change
bert-base-uncased	-3.64%	771	663	-14.01%
bert-large-uncased	-4.41%	1617	1165	-27.95%
roberta-base	-0.01%	875	673	-23.08%
distilbert-base-uncased	0.00%	812	779	-4.06%

The above two experiments use GLUE benchmark in evaluation mode. However, this does not give the true picture of the actual tradeoff between accuracy and performance unless it is fine-tuned on a specific task. Hence, we performed fine-tuning on various GLUE tasks and compared the accuracies between baseline and pruned `bert-base-uncased` model. The findings have been summarized in Table 6.4. These models were trained on Kaggle using two NVIDIA Tesla T4 15GB GPU using `max sequence length = 128`, `per device batch size = 32`, `learning rate = 2e-5` and `number of epochs = 3` (except for MRPC and WNLI where we used 5 epochs instead of 3 due to its tiny size).

Table 6.4: Comparison of accuracies between baseline and pruned `bert-base-uncased` model after fine-tuning on various GLUE tasks.

GLUE Task	Accuracy (Baseline)	Accuracy (Pruned)	% Change
SST-2	92.32	91.97	-0.38%
MRPC	84.07	81.71	-2.81%
RTE	65.70	59.57	-9.33%
WNLI	56.34	26.78	-52.46%

6.2 Question Answering

Finally, we experimented on a real-life use case by fine-tuning the `bert-base-uncased` model on SQuAD dataset to perform a question answering task. This was trained on Kag-

Figure 6.2: Training Loss vs No. of Training Steps

gle using a single NVIDIA P100 16GB GPU with `learning rate = 2e-5`, `number of training epochs = 3` and `weight decay = 0.01`. The training graph is shown in figure 6.2

An inference on the fine-tuned model yields an interesting output where the answer given by both models were exactly the same with a very small change in the accuracy scores. These findings are shown below.

```
Context: "Transformers is backed by the three most popular deep
    learning libraries - Jax, PyTorch and TensorFlow - with a
    smooth integration between them. You can easily train a model
    using one framework and switch to another for inference
    without hassle."

Question: "Which deep learning libraries back Transformers?"

Ground Truth: "Jax, PyTorch and TensorFlow"

Output (Baseline): {'score': 0.9979003071784973, 'start': 78, '
    end': 105, 'answer': 'Jax, PyTorch and TensorFlow'}

Output (Pruned): {'score': 0.9623586535453796, 'start': 76, 'end
    ': 103, 'answer': 'Jax, PyTorch and TensorFlow'}
```

Both the models gave the exact same answer "Jax, PyTorch and TensorFlow" with only 3.51% decrease in the accuracy score of pruned model.

Another inference on a bigger context and slightly difficult question is shown below.

Context: "In Grand Theft Auto V, released in 2013, a reimagined version of San Andreas appears once again, this time as a standalone island representing only the southern region of the state. The southern half of the island is dominated by Los Santos, a city that closely mirrors Los Angeles in both layout and atmosphere, while the northern portion, known as Blaine County, is less inhabited, featuring vast areas of desert, forest, and mountain, and only a few small towns. Unlike previous games, players can access the entire map from the very start without any restrictions, making it the only title in the series with a fully open world from the beginning."

Question: "Which is the lesser inhabited portion of the game?"

Ground Truth: "Blaine County"

Output (Baseline): {'score': 0.9527437090873718, 'start': 399, 'end': 412, 'answer': 'Blaine County'}

Output (Pruned): {'score': 0.9346631169319153, 'start': 399, 'end': 412, 'answer': 'Blaine County'}

As part of this experiment, we uploaded the above pretrained, pruned, and fine-tuned BERT model on Hugging Face's Model Hub, available under the model name: `22sourabh/bert-pruned-finetuned-squad`

Chapter 7

CONCLUSION AND DISCUSSION

While there is a small drop in the GLUE score, there is a significant improvement in the inference time, memory usage and total number of parameters indicating that pruning is beneficial if the model needs to be compressed to be deployed on hardware where memory and computational power is a constraint. Instead of relying on servers, this compressed model could run locally on our existing smartphones, smartwatches, and other smart gadgets.

The experiments indicates that there is a tradeoff between the accuracy and other performance metrics. Certain use cases where accuracy need not be a big issue can benefit largely from pruning the model. However, models like RoBERTa and DistilBERT have seen to be robust when it comes to the tradeoff with only little to no change in the accuracy. This indicates that there is always a scope for building robust architectures that can tackle the problem of high computational cost.

The Hugging Face library contains hundreds of transformers whose performance can be improved by pruning. This could be a scope for future research.

Pruning may not always seem beneficial e.g. The pruned BERT model failed miserably on WNLI task while it saw a very slight decrease in accuracy on SST-2. This says that which transformer to prune for with task is use case dependent.

Our experiments show that pruning attention heads significantly impacts the benchmark as compared to pruning neurons. This shows that attention mechanism contributes majorly in making decisions in the transformer.

Removing neurons without any subsequent fine-tuning can sometimes lead to little/no change in accuracy, as the network might be robust to these changes. Hence fine tuning is

necessary after every successful model pruning.

The pruned model performed very well during inference on a real-life task of question answering with only a little drop in accuracy without affecting the output indicating its practicality in several use cases.

As future research, our pruning algorithm can be further tested on multiple transformers to perform NLP tasks like Token classification, Text summarization, language translation, etc.

REFERENCES

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018.

[3] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained BERT networks. *CoRR*, abs/2007.12223, 2020.

[4] Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models, 2024.

[5] Ajay Jaiswal, Shiwei Liu, Tianlong Chen, and Zhangyang Wang. The emergence of essential sparsity in large pre-trained models: The weights that matter, 2023.

[6] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how BERT works. *CoRR*, abs/2002.12327, 2020.

[7] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[8] Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Bin Sun, Xinglin Wang, Heda Wang, and Kan Li. Turning dust into gold: Distilling complex reasoning capabilities from llms by leveraging negative data, 2023.

[9] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms, 2023.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[11] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *CoRR*, abs/1905.10650, 2019.

[12] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[13] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[14] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[15] HuggingFace. notebooks/examples/benchmark.ipynb. `https://github.com/huggingface/notebooks/blob/main/examples/benchmark.ipynb`.

[16] HuggingFace. transformers/examples/pytorch/text-classification/run_glue.py. `https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue.py`.

TURNITIN Report

Sourabh_MTECH_THESIS_Transformer_Pruning (1).pdf

ORIGINALITY REPORT

18%	12%	10%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.geeksforgeeks.org Internet Source	1%
2	huggingface.co Internet Source	1%
3	"ECAI 2020", IOS Press, 2020 Publication	1%
4	ayaka14732.github.io Internet Source	1%
5	arxiv.org Internet Source	1%
6	Submitted to Bannari Amman Institute of Technology Student Paper	<1%
7	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
8	Submitted to University of Southampton Student Paper	<1%
9	medium.com Internet Source	<1%
10	en.wikipedia.org Internet Source	<1%
11	Submitted to Indian Institute of Management, Bangalore Student Paper	<1%
12	doras.dcu.ie Internet Source	<1%

13	Sandhu, Tanvi. "Exploration of Word Embeddings With Graph-Based Context Adaptation for Enhanced Word Vectors", University of Windsor (Canada), 2024 Publication	<1 %
14	papers.nips.cc Internet Source	<1 %
15	Submitted to University of Carthage Student Paper	<1 %
16	Submitted to LTI Student Paper	<1 %
17	Submitted to Liverpool John Moores University Student Paper	<1 %
18	Submitted to University College London Student Paper	<1 %
19	muneebsa.medium.com Internet Source	<1 %
20	Submitted to Associatie K.U.Leuven Student Paper	<1 %
21	Submitted to Leeds Beckett University Student Paper	<1 %
22	"Intelligent Information and Database Systems", Springer Science and Business Media LLC, 2020 Publication	<1 %
23	Adrian David Cheok, Emma Yann Zhang. "From Turing to Transformers: A Comprehensive Review and Tutorial on the Evolution and Capabilities of Generative Models", Qeios Ltd, 2023 Publication	<1 %

Call for Paper in CICBA 2025

Springer

CCIS

IEEE Computational Intelligence Society Kolkata Chapter

SEVENTH INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE IN COMMUNICATIONS AND BUSINESS ANALYTICS

JADAVPUR UNIVERSITY, KOLKATA, INDIA

CICBA

CICBA 2025

4-6 July, 2025

(Hybrid Mode)

DEPARTMENT OF INFORMATION TECHNOLOGY, JADAVPUR UNIVERSITY, SALTLAKE CAMPUS, KOLKATA, INDIA

CCIS, Springer Scopus and DBLP Index

CICBA 2025

The Seventh conference of this series, CICBA 2025 is inviting all research findings in the field of Computational Intelligence (CI) - both Theories and Applications primarily encompassing the domains of Data Communications and Analytics. This international conference will offer a comprehensive platform for the research community to exchange their ideas on contemporary augmentation from the relevant topics.

Scope and Objective of CICBA 2025

The importance of computational intelligence (CI) is ever increasing because of its applications becoming all pervasive. Among so many application domains we, for the present, have highlighted two viz. Communications and Analytics for inviting research contributions into. This is notwithstanding other areas of application of computational intelligence. Now a days it is a great challenge to develop smart communication systems and networks that support optimized management, dynamic configuration and feasible services in communications. The research field of Computational Intelligence and cognitive radio computing are interdisciplinary and uses knowledge and methods from many areas which are very capable approaches to deal with dynamic and large-scale topology; thus, we need to explore CI-based techniques such as statistical learning, feed forward neural networks, perceptron, deep recurrent neural networks, optimizations, etc., for complicated decision making, network management, resource optimization and in-depth knowledge discovery in complex communication environments. Similarly, CI is a collection of techniques that excel to extract insights and patterns from large sets of data, and then making predictions based on that information. In fact, CI exists today that can help us to get much more value out of the data, unify that data, and for predictions about user behaviors based on it.

KEYNOTE SPEAKERS

PROF. DILIP KUMAR PRATIHAR
Department of Mechanical Engineering, Institute Chair Professor, Indian Institute of Technology Kharagpur, India

PROF. NARAYAN C. DEBNATH
School of Computing and Information Technology, Eastern International University, Vietnam

PROF. DEEPANKAR CHOUDHURY
Department of Civil Engineering, Prof. T. Kant Chair Professor (HAG), Indian Institute of Technology (IIT), Bombay,

PROF. ANKUR SINHA
Dept of Operations & Decision Sciences, Indian Institute of Management Ahmedabad Gujarat, India

PROF. ANIRBAN CHAKRABORTY
Department of Computational and Data Sciences, Indian Institute of Science Bangalore, India

DR. ARINDAM PAL
Director of Operations Research and Data Science at Optym, Sydney, Australia

Tracks (but not limited to)

- **Computational Intelligence**
  - Intelligence in Systems and Circuits
  - Intelligent Data engineering
  - Machine Learning & Deep Learning
  - Evolutionary and Swarm Intelligence
  - Fuzzy Systems and Rough Computing
  - Neural and Bio-Inspired Computing
  - Hybrid Computational Intelligence Systems
  - Computational Intelligence in Big Data and IoT
  - Natural Language Processing and Computer Vision
  - Computational Intelligence for Cybersecurity
  - Robotics and Autonomous Systems
  - Computational Intelligence for Scientific and Environmental Applications
  - Industry and Healthcare Applications
- **Data Communications and Analytics**
  - Security & Authentication
  - Smart Sensing Systems
  - Data Mining and Analytics
  - IoT Datamining
  - Block Chain Analytics
  - Network Architectures and Protocols
  - Wireless and Mobile Communication
  - IoT Communication Technologies
  - 5G and Beyond Networks
  - Edge and Fog Computing
  - Optical and Quantum Communication
  - Network Security and Privacy
  - Cloud Computing and Virtualization
  - Software-Defined Networking (SDN)
  - Internet of Things (IoT) Data Transmission
  - Blockchain for Secure Data Communication
  - Green and Sustainable Networking
  - AI-Driven Network Management
  - Cyber-Physical Systems Communication
  - High-Performance Data Analytics
  - Explainable AI in Data Analytics
  - Financial and Business Analytics
  - Climate and Environmental Data Analytics
  - Social Media and Sentiment Analysis
  - Smart Cities and Smart Grids
  - Cybersecurity Data Analytics
  - Supply Chain and Logistics Analytics
  - Retail and E-Commerce Data Analytics
  - Precision Agriculture and Smart Farming

SUBMISSION LINK OPEN

- **Chief Patron**
  Vice-Chancellor,
  Jadavpur University, Kolkata
- **Patron**
  Pro-Vice Chancellor,
  Jadavpur University, Kolkata
- **Organizing Chair**
  HoD, Dept. of Info. Tech.,
  Jadavpur University, Kolkata
- **Finance Chair**
  Finance Officer,
  Jadavpur University, Kolkata
- **Organizing Committee**
  Dr. Bibhas Chandra Dhara, JU, Kolkata
  Dr. Uttam Kumar Roy, JU, Kolkata
  Dr. Bhaskar Sardar, JU, Kolkata
  Dr. Parama Bhaumik, JU, Kolkata
  Dr. Kartick Chandra Mondal, JU, Kolkata
  Dr. Tohida Rehman, JU, Kolkata
  Dr. Munmun Bhattacharya, JU, Kolkata
  Dr. Sruti Gan Chaudhuri, JU, Kolkata
  Mr. Utpal Kumar Ray, JU, Kolkata
  Dr. Pawan Kumar Singh, JU, Kolkata
  Dr. Rohini Basak, JU, Kolkata
  Mr. Palash Kundu, JU, Kolkata
- **Program Chair**
  Dr. Kartick Chandra Mondal, JU, Kolkata
  Dr. Parama Bhaumik, JU, Kolkata
  Dr. Munmun Bhattacharya, JU, Kolkata
  Dr. Somnath Mukhopadhyay, ASU, Silchar
  Dr. Paramartha Dutta, VBU, Santiniketan
  Dr. Jotsnya Kumar Mandal, KU, Kalyani

Category	Fees
Foreign Authors Registration	$300
Industry Author Registration	Rs.7500
Academic Author Registration	Rs.6500
Students/Researcher Registration	Rs.5000
Workshop/Poster Registration	Rs.4000
Conference Only Registration	Rs.3000

WEBSITE

Publication Partner: CCIS, Springer

Paper Submission Deadline: OPEN

Acceptance Notification: May 31, 2025

Early Bird Registration Deadline: June 15, 2025

HAVE ANY QUERY +919635335745

https://www.cicba.in/ https://jadavpuruniversity.in/ https://www.ju-it.in/

Research Paper Acceptance Notification in CICBA 2025

Gmail

Sourabh Sathe <sou22600@gmail.com>

CICBA 2025 Acceptance Notification

1 message

KARTICK CHANDRA MONDAL <kartick.mondal@jadavpuruniversity.in> Sun, May 18, 2025 at 10:57 PM
Bcc: sou22600@gmail.com

Dear Author(s),

✨ **Congratulations!** ✨
Greetings from the **Organizing Committee of CICBA 2025**!

We are delighted to inform you that your manuscript has been **accepted** for a **full-paper presentation** in a **Technical Session** at the:

🎓 **Seventh International Conference on Computational Intelligence in Communications and Business Analytics (CICBA 2025)**
📍 *Jadavpur University, Kolkata, India*
🗓️ *04 – 06 July 2025*

Check your accepted paper IDs at the conference website: 🌐 Paper Acceptance Information

📝 Your submission was reviewed and accepted based on its **relevance to the conference theme**, **technical merit**, and **originality**.
You are invited to present your work in an **oral or virtual session**, as per the final schedule, which will be shared **after registration**.

Fee Payment Receipt for CICBA 2025

JADAVPUR UNIVERSITY

188 RAJA S.C. MULLICK ROAD,JADAVPUR,KOLKATA,Kolkata,WEST BANGAL, KOLKATA-700032
Date: 26-May-2025

SBCollect Reference Number :	DUO1157893
Category :	Conference-Registration-Fees-CICBA-2025
Amount :	₹6000
Name :	Sourabh Sathe
Phone No :	9561110254
Mail ID :	sou22600@gmail.com
Paper ID :	332

Registration for Full Paper Presentation in CICBA 2025

Gmail — Sourabh Sathe <sou22600@gmail.com>

CICBA 2025 Conference Registration Portal for Author and Participant
1 message

Google Forms <forms-receipts-noreply@google.com> — Mon, May 26, 2025 at 10:24 PM
To: sou22600@gmail.com

Thanks for filling out CICBA 2025 Conference Registration Portal for Author and Participant

Here's what was received.

CICBA 2025 Conference Registration Portal for Author and Participant

Your email (**sou22600@gmail.com**) was recorded when you submitted this form.

AUTHOR INFORMATION

PLEASE ENTER EACH FIELD CAREFULLY.

Enter Your Full Name *

Sourabh Sathe

Enter Your Affiliation
(University/Institute/Organization)
*

Visvesvaraya National Institute of Technology, Nagpur

Enter Your Contact Number *

9561110254

Enter Your WhatsApp Number *

9561110254

Enter Your Email ID *

sou22600@gmail.com

Enter Your Paper ID
(Enter 000 for conference only registration)
*

332

CICBA 2025 Conference Registration Portal for Author and Participant

Your submission has been successfully recorded. One email has been send to your registered gmail id.

Very soon, corresponding author will receive the article submission id to his/her e-mail after preliminary desk checking.

Submit another response